Micriμm, Inc.

New Features and Services since μC/OS-II V2.00

Jean J. Labrosse
Jean.Labrosse@Micrium.com
www.Micrium.com

Introduction

This document describes all the features and services added to µC/OS-II since the introduction of the hard cover book *MicroC/OS-II, The Real-Time Kernel*, ISBN 0-87930-543-6. The software provided with the book was version 2.00 or V2.04. The version number of the change is shown when appropriate.

New File

`OS_CFG_R.H`	(Added in V2.70)

This file is 'reference' file so that you don't have to create this file from scratch. `OS_CFG_R.H` has been added in V2.70 and is found in the '`Source`' directory of the microprocessor independent portion of µC/OS-II. It is recommended that you *copy* `OS_CFG_R.H` to `OS_CFG.H` of your project directory.

New Port File

`OS_DBG.C`	(Added in V2.62 but renamed from `OS_DEBUG.C` in V2.70)
`OS_DBG_R.C`	(Added in V2.70)

This file should be placed in the same directory as `OS_CPU_C.C`, `OS_CPU.H` and `OS_CPU_A.ASM` of the port you are using. `OS_DBG.C` defines a series of variables that are placed in ROM (code space). These variables are used by some Kernel Aware Debuggers to get information about µC/OS-II and its configuration. If you DON'T use a Kernel Aware Debugger that requires this file, you DON'T need to have it. Check you Kernel Aware Debugger documentation. `OS_DBG.C` used to be called `OS_DEBUG.C` in V2.62.

`OS_DBG_R.C` is a 'reference' file so that you don't have to create this file from scratch. `OS_DBG_R.C` has been added in V2.70 and is found in the 'Source' directory of the microprocessor independent portion of µC/OS-II.

Changes

`uCOS_II.H`	(Changed in V2.70)

This file now includes `#include` statements to include `OS_CPU.H` and `OS_CFG.H`. This allows you to compile µC/OS-II without the needs of any other library functions.

New #define Constants and Macros

`OS_ARG_CHK_EN            (OS_CFG.H, V2.04)`

This constant is used to specify whether argument checking will be performed at the beginning of most of µC/OS-II services. You should always choose to turn this feature on (when set to `1`) unless you need to get the best performance possible out of µC/OS-II or, you need to reduce code size.

`OS_CRITICAL_METHOD #3   (OS_CPU.H, V2.04)`

This constant specifies the method used to disable and enable interrupts during critical sections of code. Prior to V2.04, `OS_CRITICAL_METHOD` could be set to either `1` or `2`. In V2.04, I added a local variable (i.e. `cpu_sr`) in most function calls to save the processor status register which generally holds the state of the interrupt disable flag(s). You would then declare the two critical section macros as follows:

```
#define OS_ENTER_CRITICAL()  (cpu_sr = OSCPUSaveSR())
#define OS_EXIT_CRITICAL()   (OSCPURestoreSR(cpu_sr))
```

Note that the functions `OSCPUSaveSR()` and `OSCPURestoreSR()` would be written in assembly language and would typically be found in `OS_CPU_A.ASM` (or equivalent).

`OS_DEBUG_EN             (OS_CFG.H, V2.60)`

This constant is used to enable ROM constants used for debugging using a kernel aware debugger. The constants are found in `OS_CORE.C`.

`OS_EVENT_NAME_SIZE      (OS_CFG.H, V2.60)`

This constant determines the maximum number of characters that would be used to assign a name to either a semaphore, a mutex, a mailbox or a message queue. The name of these 'objects' would thus have to be smaller (in size) than this value. If `OS_EVENT_NAME_SIZE` is set to `0`, this feature is disabled. `OS_EVENT_NAME_SIZE` needs to accommodate a `NUL` terminated ASCII string.

`OS_FLAG_EN              (OS_CFG.H, V2.51)`

This constant is used to specify whether you will enable (when 1) code generation for the event flags.

`OS_FLAG_NAME_SIZE       (OS_CFG.H, V2.60)`

This constant determines the maximum number of characters that would be used to assign a name to an event flag group. The name of event flags would thus have to be smaller (in size) than this value. If `OS_FLAG_NAME_SIZE` is set to `0`, this feature is disabled. `OS_FLAG_NAME_SIZE` needs to accommodate a `NUL` terminated ASCII string.

`OS_FLAG_WAIT_CLR_EN     (OS_CFG.H, V2.51)`

This constant is used to enable code generation (when 1) to allow to wait on cleared event flags.

`OS_ISR_PROTO_EXT         (OS_CPU.H, V2.02)`

If you place this constant is `OS_CPU.H`, you can redefine the function prototypes for `OSCtxSw()` and `OSTickISR()`. In other words, if you add the following definition, YOU will have to declare the prototype for `OSCtxSw()` and `OSTickISR()`.

```
#define OS_ISR_PROTO_EXT  1
```

`OS_MAX_FLAGS             (OS_CFG.H, V2.51)`

This constant is used to determine how many event flags your application will support.

`OS_MEM_NAME_SIZE         (OS_CFG.H, V2.60)`

This constant determines the maximum number of characters that would be used to assign a name to a memory partition. The name of memory partitions would thus have to be smaller (in size) than this value. If `OS_MEM_NAME_SIZE` is set to `0`, this feature is disabled and no RAM is used in the `OS_MEM` for the memory partition. `OS_MEM_NAME_SIZE` needs to accommodate a `NUL` terminated ASCII string.

`OS_MUTEX_EN              (OS_CFG.H, V2.04)`

This constant is used to specify whether you will enable (when 1) code generation for mutual exclusion semaphores.

`OS_TASK_NAME_SIZE        (OS_CFG.H, V2.60)`

This constant determines the maximum number of characters that would be used to assign a name to a task. The name of tasks would thus have to be smaller (in size) than this value. If `OS_TASK_NAME_SIZE` is set to `0`, this feature is disabled and no RAM is used in the `OS_TCB` for the task name. `OS_TASK_NAME_SIZE` needs to accommodate a `NUL` terminated ASCII string.

`OS_TASK_PROFILE_EN       (OS_CFG.H, V2.60)`

This constant allows variables to be allocated in each task's `OS_TCB` that hold performance data about each task. Specifically, if `OS_TASK_PROFILE_EN` is set to `1`, each task will have a variable to keep track of the number of context switches, the task execution time, the number of bytes used by the task and more.

`OS_TASK_STAT_STK_CHK_EN (OS_CFG.H, V2.60)`

This constant allows the statistic task to determine the actual stack usage of each active task. If `OS_TASK_STAT_EN` is set to `0` (the statistic task is not enabled), you can call `OS_TaskStatStkChk()` yourself from one of your tasks. . If `OS_TASK_STAT_EN` is set to `1`, stack sizes will be determined every second.

`OS_TASK_SW_HOOK_EN       (OS_CFG.H, V2.60)`

Normally, µC/OS-II requires that you have a context switch hook function called `OSTaskSwHook()`. When set to `0`, this constant allows you to omit `OSTaskSwHook()` from your code. This configuration constant was added to reduce the amount of overhead during a context switch in applications that doesn't require the context switch hook. Of course, you will also need to remove the calls to `OSTaskSwHook()` from `OSTaskStartHighRdy()`, `OSCtxSw()` and `OSIntCtxSw()` in `OS_CPU_A.ASM`.

`OS_TICK_STEP_EN          (OS_CFG.H, V2.60)`

µC/OS-View can now 'halt' µC/OS-II's tick processing and allow you to issue 'step' commands from µC/OS-View. In other words, µC/OS-View can prevent µC/OS-II from calling `OSTimeTick()` so that timeouts and time delays are no longer processed. However, though a keystroke from µC/OS-View, you can execute a single tick at a time. If `OS_TIME_TICK_HOOK_EN` (see below) is set to `1`, `OSTimeTickHook()` is still executed at the regular tick rate in case you have time critical items to take care of in your application.

`OS_TIME_TICK_HOOK_EN     (OS_CFG.H, V2.60)`

Normally, µC/OS-II requires the presence of a function called `OSTimeTickHook()` which is called at the very beginning of the tick ISR. When set to `0`, this constant allows you to omit `OSTimeTickHook()` from your code. This configuration constant was added to reduce the amount of overhead during a tick ISR in applications that doesn't require this hook.

The following table summarizes some of the new `#define` constants in `OS_CFG.H` which were all added in since V2.00.

`#define` name in `OS_CFG.H`	**... to enable the function:**
`OS_DEBUG_EN`	Enable debug constants in `OS_CORE.C`. If you are using a kernel aware debugger, you should enable this feature.

`OS_EVENT_NAME_SIZE`	`OSEventNameGet()`
	`OSEventNameSet()`
	And, to allow naming semaphores, mutexes, mailboxes and message queues.

`OS_FLAG_ACCEPT_EN`	`OSFlagAccept()`
`OS_FLAG_DEL_EN`	`OSFlagDel()`

`OS_FLAG_NAME_SIZE`	`OSFlagNameGet()`
	`OSFlagNameSet()`
	And, to allow naming event flag groups.

`OS_FLAG_QUERY_EN`	`OSFlagQuery()`
`OS_MBOX_ACCEPT_EN`	`OSMboxAccept()`
`OS_MBOX_DEL_EN`	`OSMboxDel()`
`OS_MBOX_POST_EN`	`OSMboxPost()`
`OS_MBOX_POST_OPT_EN`	`OSMboxPostOpt()`
`OS_MBOX_QUERY_EN`	`OSMBoxQuery()`

`OS_MEM_NAME_SIZE`	`OSMemNameGet()`
	`OSMemNameSet()`

`OS_MEM_QUERY_EN`	`OSMemQuery()`
`OS_MUTEX_ACCEPT_EN`	`OSMutexAccept()`
`OS_MUTEX_DEL_EN`	`OSMutexDel()`
`OS_MUTEX_QUERY_EN`	`OSMutexQuery()`
`OS_Q_ACCEPT_EN`	`OSQAccept()`

OS_Q_DEL_EN	OSQDel()
OS_Q_FLUSH_EN	OSQFlush()
OS_Q_POST_EN	OSQPost()
OS_Q_POST_FRONT_EN	OSQPostFront()
OS_Q_POST_OPT_EN	OSQPostOpt()
OS_Q_QUERY_EN	OSQQuery()
OS_SEM_ACCEPT_EN	OSSemAccept()
OS_SEM_DEL_EN	OSSemDel()
OS_SEM_QUERY_EN	OSSemQuery()
OS_SEM_SET_EN	OSSemSet()

OS_TASK_NAME_SIZE	OSTaskNameGet()
	OSTaskNameSet()
	And, to allow naming tasks.

OS_TASK_PROFILE_EN	To allocate variables in OS_TCB for performance monitoring of each task at run-time.
OS_TASK_QUERY_EN	OSTaskQuery()
OS_TASK_STAT_STK_CHK_EN	OS_TaskStatStkChk()
OS_TASK_SW_HOOK_EN	OSTaskSwHook()
OS_TICK_STEP_EN	To support the stepping feature of µC/OS-View.
OS_TIME_DLY_HMSM_EN	OSTimeDlyHMSM()
OS_TIME_DLY_RESUME_EN	OSTimeDlyResume()
OS_TIME_GET_SET_EN	OSTimeGet() and OSTimeSet()
OS_TIME_TICK_HOOK_EN	OSTimeTickHook()
OS_SCHED_LOCK_EN	OSSchedLock()and OSSchedUnlock()

New Data Types

`OS_CPU_SR                (OS_CPU.H, V2.04)`

This data type is used to specify the size of the CPU status register which is used in conjunction with `OS_CRITICAL_METHOD` #3 (see above). For example, if the CPU status register is 16-bit wide then you would `typedef` accordingly.

`OS_FLAGS                 (OS_CFG.H, V2.51)`

This data type determines how many bits an event flag group will have. You can thus `typedef` this data type to either `INT8U`, `INT16U` or `INT32U` to give event flags either 8, 16 or 32 bits, respectively.

New Hook Functions

`void OSInitHookBegin(void)          (OS_CPU.C, V2.04)`

This function is called at the very beginning of `OSInit()` to allow for port specific initialization BEFORE µC/OS-II gets initialized.

`void OSInitHookEnd(void)            (OS_CPU.C, V2.04)`

This function is called at the end of `OSInit()` to allow for port specific initialization AFTER µC/OS-II gets initialized.

`void OSTCBInitHook(OS_TCB *ptcb)    (OS_CPU.C, V2.04)`

This function is called by `OSTCBInit()` during initialization of the TCB assigned to a newly created task. It allows port specific initialization of the TCB.

`void OSTaskIdleHook(void)           (OS_CPU.C, V2.51)`

This function is called by `OSTaskIdle()`. This allows you to STOP the CPU and thus reduce power consumption while there is nothing to do.

New Functions

The following table provides a list of all the new functions (i.e. services) that YOUR application can call. The list also includes functions that used to exist but, if these are in this list, it's because their API may have changed.

Refer to the ***Reference Manual*** of the current release for a description of these functions.

Function Name	**File**	**Enabled By …**
`OSEventNameGet()`	`OS_CORE.C`	`OS_EVENT_NAME_SIZE`
`OSEventNameSet()`	`OS_CORE.C`	`OS_EVENT_NAME_SIZE`
`OSFlagAccept()`	`OS_FLAG.C`	`OS_FLAG_EN && OS_FLAG_ACCEPT_EN`
`OSFlagCreate()`	`OS_FLAG.C`	`OS_FLAG_EN`
`OSFlagDel()`	`OS_FLAG.C`	`OS_FLAG_EN && OS_FLAG_DEL_EN`
`OSFlagNameGet()`	`OS_FLAG.C`	`OS_FLAG_NAME_SIZE`
`OSFlagNameSet()`	`OS_FLAG.C`	`OS_FLAG_NAME_SIZE`
`OSFlagPend()`	`OS_FLAG.C`	`OS_FLAG_EN`
`OSFlagPendGetFlagsRdy()`	`OS_FLAG.C`	`OS_FLAG_EN`
`OSFlagPost()`	`OS_FLAG.C`	`OS_FLAG_EN`
`OSFlagQuery()`	`OS_FLAG.C`	`OS_FLAG_EN`
`OSMboxDel()`	`OS_MBOX.C`	`OS_MBOX_EN && OS_MBOX_DEL_EN`
`OSMboxPostOpt()`	`OS_MBOX.C`	`OS_MBOX_EN && OS_MBOX_POST_OPT_EN`
`OSMutexAccept()`	`OS_MUTEX.C`	`OS_MUTEX_EN && OS_MUTEX_ACCEPT_EN`
`OSMutexCreate()`	`OS_MUTEX.C`	`OS_MUTEX_EN`
`OSMutexDel()`	`OS_MUTEX.C`	`OS_MUTEX_EN && OS_MUTEX_DEL_EN`
`OSMutexPend()`	`OS_MUTEX.C`	`OS_MUTEX_EN`
`OSMutexPost()`	`OS_MUTEX.C`	`OS_MUTEX_EN`
`OSMutexQuery()`	`OS_MUTEX.C`	`OS_MUTEX_EN && OS_MUTEX_QUERY_EN`
`OSQAccept()`	`OS_Q.C`	`OS_Q_EN && OS_Q_ACCEPT_EN`
`OSQDel()`	`OS_Q.C`	`OS_Q_EN && OS_Q_DEL_EN`
`OSQFlush()`	`OS_Q.C`	`OS_Q_EN && OS_Q_FLUSH_EN`
`OSQPend()`	`OS_Q.C`	`OS_Q_EN`
`OSQPost()`	`OS_Q.C`	`OS_Q_EN`
`OSQPostFront()`	`OS_Q.C`	`OS_Q_EN && OS_Q_POST_FRONT_EN`
`OSQPostOpt()`	`OS_Q.C`	`OS_Q_EN && OS_Q_POST_OPT_EN`
`OSSemDel()`	`OS_SEM.C`	`OS_SEM_EN && OS_SEM_DEL_EN`
`OSSemSet()`	`OS_SEM.C`	`OS_SEM_EN && OS_SEM_SET_EN`
`OSTaskNameGet()`	`OS_TASK.C`	`OS_TASK_NAME_SIZE`
`OSTaskNameSet()`	`OS_TASK.C`	`OS_TASK_NAME_SIZE`

References

µC/OS-II, The Real-Time Kerne, 2nd Editionl
Jean J. Labrosse
CMP Books, 2002
ISBN 1-57820-103-9

Contacts

Micriµm, Inc.
949 Crestview Circle
Weston, FL 33327
954-217-2036
954-217-2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com
WEB: www.Micrium.com

CMP Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
(785) 841-1631
(785) 841-2624 (FAX)
WEB: http://www.cmpbooks.com
e-mail: rdorders@cmpbooks.com